◎博士论坛◎

# Bitmap 结构在高性能网络算法设计中的应用

杨保华[1,2],亓亚烜[2,3],薛一波[2,3],李 军[2,3]
YANG Bao-hua[1,2],QI Ya-xuan[2,3],XUE Yi-bo[2,3],LI Jun[2,3]

1.清华大学 自动化系,北京 100084
2.清华大学 信息技术研究院,北京 100084
3.清华信息科学与技术国家实验室,北京 100084
1.Department of Automation,Tsinghua University,Beijing 100084,China
2.Research Institute of Information Technology,Tsinghua University,Beijing 100084,China
3.National Laboratory for Information Science and Technology,Tsinghua University,Beijing 100084,China
E-mail:ybh07@mails.tsinghua.edu.cn

YANG Bao-hua,QI Ya-xuan,XUE Yi-bo,et al.Bitmap data structure:Towards high-performance network algorithms designing.Computer Engineering and Applications,2009,45(15):1-5.

**Abstract**: Bitmap is an efficient data compression technology for linear storage structures.It has been used in various cases to improve the performance of network processing.This paper proves the effectiveness of bitmap technology in storage requirement reduction for data structures of existing algorithms.A mathematical model is provided to illustrate the benefits of the bitmap technology,and experiment results also demonstrates the compression ability of bitmap technology.This paper also analyzes the advantages and disadvantages of using the bitmap technology,to indicate that bitmap,as an efficient technology for improving storage performance,provides an implicational guideline for future algorithms designing.

**Key words**: Bitmap;route lookup;packet classification;pattern matching;high performance network processing

**摘 要**:基于 Bitmap 数据结构的数据压缩技术是一种针对线性存储结构的有效压缩方法,虽被广泛用于网络处理的多个领域(路由查找、网包分类等),却一直缺乏深入的分析。给出了 Bitmap 结构能提高算法空间性能的理论根据。总结了 Bitmap 结构在典型网络处理算法中的各种应用,给出了 Bitmap 结构的数学模型,并通过实例分析了 Bitmap 结构的优势和不足。Bitmap 技术是一种能有效改善网络处理算法存储空间性能的通用技术,并给未来高性能网络处理算法设计提出以及现有算法的改进都提供了启发思路。

**关键词**:Bitmap;路由查找;网包分类;模式匹配;高性能网络处理
**DOI**:10.3778/j.issn.1002-8331.2009.15.001 **文章编号**:1002-8331(2009)15-0001-05 **文献标识码**:A **中图分类号**:TP183

现代社会,Internet 在人们工作和生活中的应用越来越广泛,人们对于网络性能特别安全性能的需求越来越高。由于链路技术(如光纤技术)的飞速进步,使得 Internet 上的带宽迅速增长,对主干网的路由器就提出了更高的要求。同样,高速的网络流量对网关防火墙和入侵监测系统(Intrusion Detection System,IDS)、入侵防御系统(Intrusion Prevention System,IPS)等安全设备构成了很大的压力。而上述设备中的网络处理算法则直接决定设备的性能。

上述几种不同应用环境下的网络处理算法往往需要同时具备较高的时间性能(处理网包速度要快)和空间性能(数据结构存储空间要小)。近些年来,虽然不断有更高性能的算法被提出,但是多数是在原有经典算法基础上进行改进。由于网络过滤问题内在的复杂性因素[1],单纯从数学复杂度上提高算法的处理速度已经难有作为[2]。但是,结合不断更新的硬件特性,如大容量的缓存(cache),高速片上内存(on-chip memory)等硬件特性,并改进算法的数据结构,往往可以获得较高的性能提高。Bitmap 技术就是一种典型的数据结构压缩技术,广泛用于经典的路由查找、网包分类和模式匹配算法的改进,本文将对 Bitmap 技术的背景、复杂度以及应用做详细介绍,并通过实验结果来说明 Bitmap 对算法性能提升的有效性,最后提出Bitmap

基金项目:国家高技术研究发展计划(863)(the National High-Tech Research and Development Plan of China under Grant No.2007AA01Z468)。
作者简介:杨保华(1986-),男,博士生,主要研究领域为网络安全;亓亚烜(1979-),男,博士生,主要研究领域为网包分类算法和模式匹配算法;薛一波(1967-),男,博士,研究员,主要研究领域为网络与信息安全、计算机体系结构、并行处理等;李军(1962-),男,博士,研究员,主要研究领域为网络安全、模式识别和信号处理领域。
收稿日期:2008-12-08 修回日期:2009-01-22

数据结构在网络处理算法中进一步应用的方向。

## 1 网络处理中的典型问题

为了阐明 Bitmap 数据结构在网络处理算法中的应用，首先介绍三类典型的网络处理问题：路由查找、网包分类和模式匹配。

这三个问题在数学上均可以归结为不同维度的搜索空间中的点匹配问题。具体说来，路由查找是在目的 IP 地址一维空间中进行最长前缀的匹配；网包分类是在源地址、目的地址、源端口、目的端口和协议域五维空间中进行最高优先级匹配；模式匹配算法是在文本空间中找出可能出现的模式串进行字符串匹配。因此，上面三种算法均需要用到查找、匹配等环节。

路由查找算法多采用的数据结构为 trie 结构。在目的地址上构建一棵地址 trie，路由查找问题也就变成了查找某个叶子节点的问题。有代表性的路由查找算法像 LC-trie 算法[3]、Lulea 算法[4]均采用了 trie 的数据结构。其结构的特点是随着层数的增加节点数将迅速增长。

网包分类代表算法 HiCuts 算法[5]可以获得较好的时间性能和空间性能的均衡，也采用了 trie 结构。由于需要在长达 104 位的 5 个域上[1]进行分割，往往需要大量的存储空间。

模式匹配问题在 IDS、IPS 设备中一直是性能的瓶颈。目前网络入侵特征库、病毒特征库规模已超过十万条[6]，因此，多模匹配算法一直是 IDS、IPS 设备以及杀毒软件的核心技术。例如经典多模匹配算法 AC 算法[7]采用状态机的数据结构，要维持大量的存储信息。

无论是路由器这样的网络转发核心设备还是防火墙、IDS、IPS 这样的网关安全核心设备，处理算法都要维持相当可观的存储信息。当超过缓存或高速内存（SRAM）容量时，就需要将这些信息保存到低速存储器（DRAM）中，从而使读取速度成为性能的瓶颈。因此，改进数据结构的存储性能，将能有效地改进算法性能。

## 2 Bitmap 技术

Bitmap 技术定义如下：

定义 1（Bitmap 技术） 利用“Bitmap Array”辅助数据结构，将“Origin Array”中存在的“Redundant section”用单个有效单元存储，来消除“Origin Array”中的冗余信息，以实现数据结构无损压缩的技术，称为 Bitmap 压缩技术。

### 2.1 模型

假设 Origin Array 单元个数为 $n$，一个单元需要的存储比特数为 $usize$，即每个单元需要计算机的 $usize$ 个比特来存储。Origin Array 需要的总存储为 $n\times usize$ bit。若 Origin Array 中 Redundant section 的个数为 $m$，则可知 Compressed Array 的单元个数为 $m$，需要总的存储为 $m\times usize$ bit。Bitmap Array 中单元个数跟 Origin Array 相同，每个单元大小为 1 bit，需要总的存储为 $n$ bit。

采用 Bitmap 思想进行压缩后空间跟原空间的比值，即压缩系数 $\phi$，计算如下：

$$\phi=\frac{n+m\cdot usize}{n\cdot usize}=\frac{1}{usize}+\frac{m}{n}=h+i \tag{1}$$

其中，$h=1/usize$ 为微度系数，反映了数据结构中单元的“微小”程度，$h$ 越大，则 Origin Array 中的单元越小，Origin Array 就越“细微”。在一般数据结构中，最小存储单元为 1 字节（8 bit），从而一般有 $h\leqslant 1/8$。$i=m/n$ 为散度系数，是 Origin Array 中 Redundant section 个数与数据单元个数的比例，反映了在 Bitmap 压缩时数据块分布的离散程度。一般地，在确定的前提下，Origin Array 中存储数据的连续重复性越好，则 Redundant section 的个数越少，采用 Bitmap 进行压缩后的效果也就越好；反之，Redundant section 的个数越多，说明 Origin Array 中数据连续重复性越差，压缩效果也就越差。

### 2.2 Bitmap 计算

采用 Bitmap 压缩思想可以将 Origin Array 压缩为更小的结构 Compressed Array 和 Bitmap Arrays（图 1），但是这样也带来了时间上的代价。原来要取得 Origin Array 中第 $k$ 个单元的值，直接访问 Origin Array[$k$]即可，而利用压缩后的结构 Compressed Array 和 Bitmap Array，则首先要找到 Origin Array 中第 $k$ 个特定单元对应到 Compressed Array 中单元的序号 $k'$，然后利用 $k'$ 得到 Compressed Array[$k'$]的值。在这个过程中，需要利用 Bitmap Array 来通过 $k$ 计算出 $k'$。

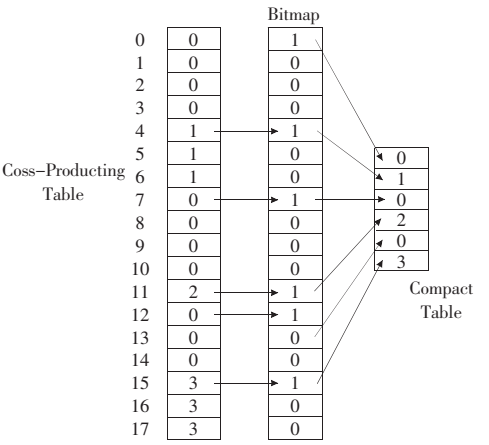


图 1 Bitmap 技术压缩叉积表

求解 $k'$ 时需要计算比特串中 1 的个数。简单的蛮力算法逐单元按照顺序统计 Bitmap Array 中的值为 1 的单元个数。该算法的时间复杂度是 $O(i)$，最坏情况下要做 $n-1$ 次加法运算，平均要 $n/2$ 次加法运算。改进的计算方法包括软件算法和硬件算法两种思路。

#### 2.2.1 软件算法

Lulea 算法[4]通过预处理操作，利用额外的数组来保存 Bitmap 中每隔若干位的 PopCount 值，来加快处理时的速度（图 2）。Lulea 算法在最初的设计中计算长度为 $2^{16}$ 的 Bitmap 中 PopCount 问题，每 64 位作为一个高级单元组，这样每个高级单

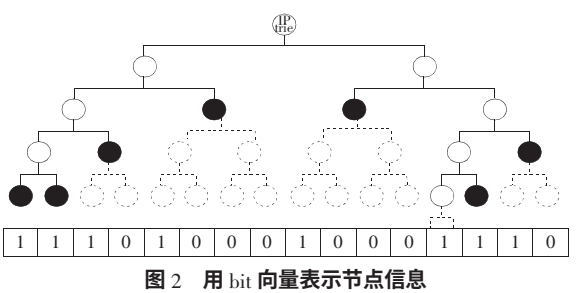


图 2 用 bit 向量表示节点信息

[1] 文中指 IPv4 协议，5 个域包括源地址（32 bit）、目的地址（32 bit）、源端口（16 bit）、目的端口（16 bit）、协议（8 bit）。

元组由 4 个次级单元组组成(即每 16 位作为一个次级单元组)。这样在每个高级单元组的开头计算此前所有位中 1 的个数并保存在 Base index array 中,在每个次级单元组的开头计算所属高级单元组中此前所有位中 1 的个数,并和跳转的指针一起保存在 Code word array 中。容易证明存在如下定理。

**定理 1** 设 Lulea 算法中高级单元组大小为 $a$,次级单元组大小为 $b$,所求 Bitmap 串长度为 $n$($n>b$),则所需要的相加次数为:

$$\begin{cases} 0 & \text{if } n\%a=0 \\ 1+n\%b & \text{otherwise} \end{cases}$$

#### 2.2.2 硬件算法

软件算法在解决 PopCount 问题时都会带来过多的计算或存储代价。基于硬件平台的内嵌指令提供了另外一种解决的思路。

多数 NP 平台(如 IXP28XX)自带能直接统计 Bitmap 中值为 1 单元个数的指令 PopCount。该平台计算一个 32 位 bit 串仅需 3 个系统周期[8],时间复杂度为 O(1)。对比目前 IA 架构和 NP 架构的内存读取需要几十到几百个系统周期来说,代价可以忽略。图 3 是 Bitmap 的长度为 256 时,分别采用蛮算、Lulea 方法和 PopCount 指令计算前 $m$ 位中 1 的个数时需要的计算次数(即加运算次数)。可以看出,PopCount 指令对长度为 256 的 Bitmap 的 PopCount 计算平均需要 3.5 次,最坏需要 7 次,运算次数为蛮算算法(最坏 255 次)的 2.7%左右。

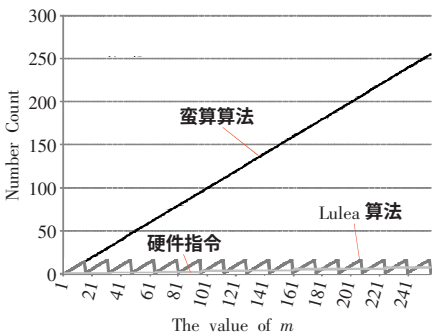


图 3 三种方式计算 PopCount

可以看出,硬件指令解决 PopCount 问题无疑具有最好的时间性能,而且没有引入额外的存储代价。但是这需要具有 POPCOUNT 特殊指令的硬件平台支持。

## 3 应用实现

从前面的分析可以看出,Bitmap 能很好地改进算法的空间性能。本章通过 Bitmap 在具体网络处理算法中的应用,来总结 Bitmap 的实际性能改进效果。

### 3.1 路由查找算法中的应用

#### 3.1.1 Lulea 算法中的应用

路由查找的目的在于找到转发表(forwarding table)中跟网包头部中目的地址匹配且具有最长前缀的表项,然后根据对应的下一跳信息转发该网包,在路由器上实现千兆带宽的实时转发需要超过每秒百万次的的查找操作。首个在通用处理器上实现达到 Gb/s 速度实时路由查找的设计是由 Lulea 科技大学的 Mikael Degermark 等人在 1997 年提出设计[4],其最重要的优化技术就是利用了 Bitmap 进行表项存储的压缩。

Mikael Degermark 等人的设计采用了树结构来表示 IP 地址 32 位结构,并采用 3 层分级结构进行压缩。在三级结构中采用位向量来表示节点信息,在 bit 向量上采用 Bitmap 压缩存储。用 1 表示某条新表项的开始,0 表示此点上对应表项跟前继节点是相同的。以第一级结构做例子进行分析,原指针的空间为 $2^{16}$,每个指针大小为 16 位;以每个指针作为压缩单元进行压缩,可得微度系数 $h=\frac{1}{usize}=\frac{1}{16}$。Lulea 算法的作者通过调研指出当时大多数路由表中虽然规则很多,但不同的转发目的地址却很少。例如一个 40K 条规则的路由表仅仅有不到 60 个不同的转发目的地址[4]。因此,散度系数 $i=\frac{m}{n}$ 也会较小。从而采用 Bitmap 能大大提高算法的空间性能。

经过 Bitmap 压缩,forwarding 表被大大缩小了。具有40 000 条路由表项的路由表仅有 150~160 KB 大小,可以放进 Pentium Pro 系列的二级 cache 中[4],从而很好地提高查找速度。

#### 3.1.2 Tree Bitmap 算法中的应用

Tree Bitmap 算法[9]由 Cisco 系统公司的 Will Eatherton 等人提出,是目前应用较为广泛的一种利用 bitmap 结构的路由查找算法。通过利用 bitmap 思想改进 Multibit Tree 的数据结构,有效对存储空间进行压缩。

Tree Bitmap 算法将每三层的 tree 节点作为一个结点单元,这样一次可以处理 3 bit 的 prefix 信息。与 Lulea 算法不同的是,Tree Bitmap 算法设计上采用了两个 bitmap:Internal bitmap 和 extending bitmap。

如图 4 所示,第一级的 3 层节点被作为一个结点单元,其 Internal bitmap(1-00-0110)中的"1"表明了三层结构上哪些节点有匹配的前缀规则;而 extending bitmap(11000011)中的"1"则标志本单元的底层节点中哪些是有效的(指向其他子级结点单元)。

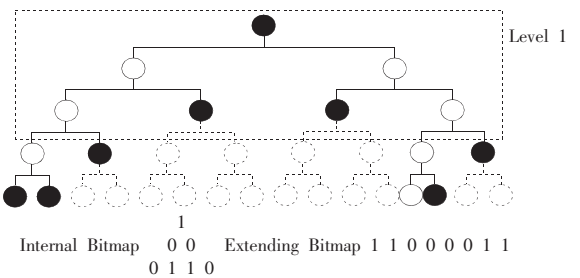


图 4 采用 Bitmap 设计的 Tree Bitmap 算法

实验表明,Tree Bitmap 结构具有很好的空间压缩性能,对于 Mae-East 规则集(40 902 条前缀规则),tree 共有 28 216 个结点,需要大约 452 K 存储空间。

### 3.2 网包分类算法中的应用

#### 3.2.1 改进 RFC 算法

RFC 算法[10]是目前网包分类算法中时间性能最好的算法之一,但是存在着数据结构占用内存大的缺点。中国科技大学的 Liu Duo 等人利用 Bitmap 技术对 RFC 算法进行了改进[8],对 RFC 算法占据大量内存的叉积表(Cross-Producting Table[10])进

行压缩存储，并利用 IXP2800 的 PopCount 指令优化 PopCount 的计算时间，获得了超过 10 Gb/s 的网包分类速度。

RFC 算法中的叉积表可被视为 Origin Array，而 Element Array 则可对应到 Compressed Array。实现的时候将 Bitmap Array 和 Element Array 存放到一起，利用 IXP2800 可以一次读入连续 64 Byte 的指令，可以执行一次读取操作就读入 Bitmap Array 和对应的 Element Array。而利用 IXP2800 特有的 PopCount 指令，可以快速地通过 Bitmap Array 获得对应单元在 Compressed Array 中的序号。

通过表 1 中可以看到，对于大于 5 700 条的规则，原始 RFC 算法数据结构不能放到 IXP2800 的 SRAM 中，改进的算法则可以实现，数据结构不用放到低速的 DRAM 中，这意味着算法性能的提高。

表 1　RFC 算法采用 Bitmap 压缩后的内存需求减少比较[8]

| Num. of Rules | Memory Requirement of CPT and CCPT/MB | | | | | | Total Memory Requirement/MB | |
|---|---|---|---|---|---|---|---|---|
| | $X$ | $X'$ | $Y$ | $Y'$ | $Z$ | $Z'$ | RFC | Bitmap-RFC |
| 5 700 | 59.2 | 18.5 | 25.8 | 8.1 | 64.1 | 16.0 | 149.9 | 43.4 |
| 8 050 | 80.7 | 25.2 | 64.3 | 20.1 | 127.7 | 39.9 | 273.5 | 86.0 |
| 12 K | 106.5 | 33.3 | 106.3 | 33.2 | 284.9 | 71.2 | 498.6 | 138.5 |
| 17 K | 191.0 | 59.7 | 185.0 | 57.8 | 570.7 | 178.4 | 947.6 | 296.7 |

### 3.2.2　改进 HiCuts 算法

HiCuts 算法[5]同样是网包分类算法中性能最好的算法之一。通过启发式的决策函数，HiCuts 算法能获得较好的时间性能和空间性能的均衡。但是 HiCuts 算法无法保证最坏的时间性能和空间性能。清华大学的 Qi Yaxuan 等人提出了基于 HiCuts 算法的改进算法——ExpCuts 算法[11]。通过确定 HiCuts 算法每步切分参数保证了时间性能，并运用 Bitmap 压缩提高了算法的空间性能，实现了同时在时间和空间性能上的优化。

HiCuts 算法采用了树状的数据结构来存储五元组[2]决策信息，每个节点都有多个后继节点，但由于规则有限，实际上这些节点的后继有很多都是空指针，因此可以采用压缩冗余后继节点信息的方式来优化节点的信息存储。采用了 CPA+ABS（Compressed Pointer Array+Aggregation Bit String）的方式，在 IXP2850 平台上利用 Bitmap 技术实现了对决策树的压缩。

如图 5 所示，CPA 可被视为 Compressed Array，而 HABS 相当于 Bitmap Array。值得注意的是，为避免硬件平台 IXP2850 的额外的内存读取次数，将 Cut 的维度、位置和 HABS 等信息放到了一个 Word 中。这样，HABS 的长度为 16 bit，因此在压缩节点的时候，将每 16 个节点作为一个压缩单元进行压缩，这样进一步减小了微度系数 $h=1/usize=1/256$。但是增大压缩单元尺寸，增大了散度系数 $i$，这样的压缩效果一般来说是比单节点之间压缩要差一些。

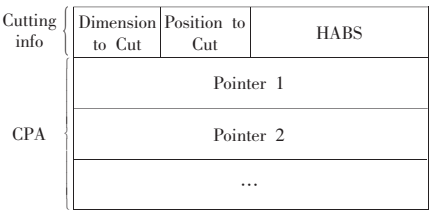


图 5　ExpCuts 节点单元的压缩存储结构

同样，ExpCuts 算法也采用了 IXP2850 的 PopCount 指令来计算 PopCount，实验结果[11]表明，ExpCuts 在保证了极高时间性能的前提下，跟不经压缩的 HiCuts 算法相比，内存需求降到了原来的 15%左右（图 6、7）。

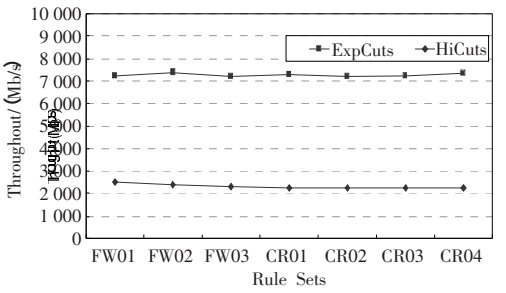


图 6　ExpCuts、HiCuts 算法吞吐性能比较[11]

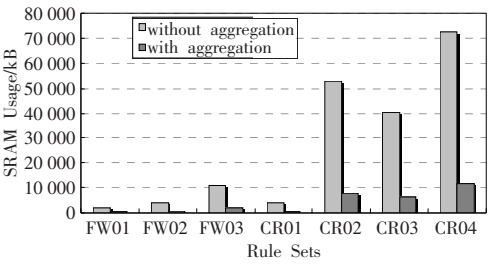


图 7　内存需求对比[11]

## 3.3　模式匹配算法中 AC 算法的改进

AC 算法[7]将规则集构建为自动状态机以实现多模式串同时匹配，是多模匹配算法的经典算法之一。但当规则集的字母表比较大时（比如 ASCII 表 256 个字符），由于要维持较多的失败跳转，AC 算法需要占用很大的存储。加利福尼亚大学的 Nathan Tuck 等人将 Bitmap 思想利用到了 AC 算法中，通过状态压缩和路径压缩，获得了极大的空间性能改进[12]。

Tuck 等人观察到非确定状态机（NFA）中每个节点的后继跳转表中有大量的失败跳转，为了降低存储，通过用一个 Bitmap 来表示某个状态跳转是否有效，虽然引入了额外的 Bitmap 存储代价，但是避免了大量跳转信息重复的存储。图 8 展示了这样一个压缩存储的节点跟原始节点存储的对比。对于字母表{ABCD}、bitmap=0101 表明，只有跳转状态 B 和 D 是有效的，这样大量的无效状态只需要保存一个失败的指针即可，

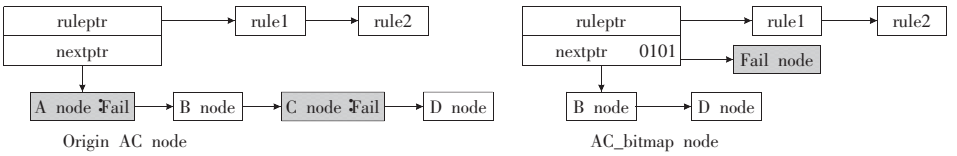


图 8　采用 Bitmap 压缩的 AC 状态机节点存储跟原始节点存储比较示意

[2] 源地址、目的地址、源端口、目的端口、采取行动

而不像原始算法那样每个无效状态都要保存一个失败的指针。Tuck 等人在 snort 规则集[13]上的实验表明，通过引入 Bitmap 避免无效存储，算法的空间性能提高到原算法的 23 倍左右。

而不对确定状态机（DFA）进行压缩，是由于确定状态机中不保存失败信息。基于 snort 规则集构建的状态机表明，出现下一跳信息中连续重复状态较少，散度系数 $i$ 变大，压缩效果变差。对 DFA 进行 Bitmap 压缩的实验获得了 2.5 倍左右的空间性能提高。

## 4 结论

### 4.1 Bitmap 技术优势

算法的设计，往往与算法所采用的数据结构关系密切。合适的数据结构能充分发挥算法的性能，反之不恰当的数据结构则会导致算法实际性能的下降。Bitmap 设计之所以能提高众多的网络处理算法性能，跟对网络处理算法设计所采用数据结构的特殊性观察密不可分。

实际的网络处理中，规则数量往往较多，单条规则包含多个域，往往需要较大的存储空间。而为了提高处理速度，实现线速处理，大量采用线性结构进行存储，利用线性快速索引特性来保证时间性能。然而线性结构连续存储的特点往往会造成存储效率不高：为了保证顺序访问的连续，即使对于空单元也要分配存储空间。通过对网络实际处理中的观察，这种存储资源浪费在网络算法中是不可忽视的。而 Bitmap 设计能在保证访问速度的前提下有效的提高了存储效率。同时由于 Cache 机制的存在，Bitmap 在压缩了算法的数据结构后，能有效的提高高速存储区的命中率，反过来实现时间性能的极大提高。

实验结果表明：无论在路由查找、网包分类还是模式匹配算法的改进中，Bitmap 设计表现都是让人满意的。另外，Bitmap 结构不影响算法自身的设计，作为一种改善存储的设计，可以很容易的融合到现有算法中，通过编写专用的处理模块，可以方便的实现嵌入。

### 4.2 Bitmap 存在不足

Bitmap 设计能有效的提高网络处理算法的实际处理性能，但是从 Bitmap 设计本身出发，也存在一些可以改进的地方。首先是算法性能跟数据分布相关。Bitmap 的优点在于能压缩数组中不需要的部分信息，从而提高空间性能，但是，这种性能的提高跟数组中单元的分布情况是有很大关系的。为了更好地提升性能，在采用 Bitmap 设计时可以对存储的数据进行散度评估，散度系数较大的数据结构采用直接线性存储，反之采用 Bitmap 进行压缩存储。此外，还可以考虑划分数据集为多个数据子集，使得每个子集的数据存储结构散度系数都较小。

## 5 展望

随着计算机技术的发展和网络技术的不断成熟，对于网络处理设备，特别是安全设备提出了越来越高的性能要求。网络处理算法作为设备的核心技术，很大程度上决定着设备的性能。由于硬件设计的特殊性，如何将算法实现的数据结构和硬件设计结合在一起、发挥出两者共同的长处，成为了下一步实现高性能处理设备的重要途径。可以预期，只有充分结合硬件特点来设计高效的算法，才能满足对于高性能处理设计的需求。

## 参考文献：


[1] Gupta P，McKeown N.Algorithms for packet classification[J].IEEE Network，2001，15(2)：24-32.

[2] Overmars M H，van der Stappen A F.Range searching and point location among fat objects[J].Journal of Algorithms，1996，21(3)：629-656.

[3] Nilsson S，Karlsson G.IP-address lookup using LC-tries[J].IEEE Journal on Selected Areas in Communications，1999，17(6)：1083-1092.

[4] Degermark M.Small forwarding tables for fast routing lookups[C]//ACM SIGCOMM Computer Communication Review，1997，27(4)：3-14.

[5] Gupta P，McKeown N.Packet classification using hierarchical intelligent cuttings[J].IEEE Micro，2000，20(1)：34-41.

[6] Total number of signatures[EB/OL].(2007).http://www.clamAV.org.

[7] Aho A V，Corasick M J.Efficient string matching：an aid to bibliographic search[C]//Communications of the ACM，1975，18(6)：333-340.

[8] Duo L.High-performance packet classification algorithm for many-core and multithreaded network processor[C]//Proceedings of the 2006 International Conference on Compilers，Architecture and Synthesis for Embedded Systems.Seoul，Korea，2006.

[9] Eatherton W，Varghese G，Dittia Z.Tree bitmap：hardware/software IP lookups with incremental updates[C]//ACM SIGCOMM Computer Communication Review，2004，34(2)：97-122.

[10] Pankaj G，Nick M.Packet classification on multiple fields[C]//Proceedings of the Conference on Applications，Technologies，Architectures，and Protocols for Computer Communication.Cambridge，Massachusetts，United States：ACM，1999.

[11] Yaxuan Q.Towards optimized packet classification algorithms for multi-core network processor[C]//Proceedings of the 2007 International Conference on Parallel Processing(ICPP)，Greece，2007.

[12] Tuck N.Deterministic memory-efficient string matching algorithms for intrusion detection[C]//Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies，HongKong，China，2004.

[13] Sourcefire，Inc.Sourcefire VRT Certified Rules[EB/OL].(2007).http://www.snort.org.

[14] CaviumNetworks，IncOCTEON™ CN38XX/CN36XX Multi-Core MIPS64 Based SoC Processors 2007[EB/OL].(2007).http://www.caviumnetworks.com.